SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

SENSORIRAJAPINNAN INTEGRAATIO SAVONIAN SAMI-JÄRJESTELMÄÄN

TEKIJÄ/T: Joona Myllys

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Joona Myllys	
Työn nimi Sensorirajapinnan integraatio Savonian SaMi-järjestelmään.	
Päiväys 10.05.2018	Sivumäärä/Liitteet 26/0
Ohjaaja(t) Mikko Pääkkönen, TKI-asiantuntija, Keijo Kuosmanen, Lehtori	
Toimeksiantaja/Yhteistyökumppani(t) Mikko Pääkkönen, Savonia Ammattikorkeakoulu	

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutustua johonkin sensorirajapintaan ja toteuttaa tämän perusteella esimerkkisovellus toteuttaen tätä rajapintaa. Rajapintavaihtoehtoja oli lähtötilanteessa kaksi, joista toinen valikoitui järkevämmäksi vaihtoehdoksi, ollessaan huomattavasti modernimpi ja tukiessaan suurempaa määrää eri data formaatteja.

Työssä käydään läpi, minkälainen tietokantarakenne toteutettiin sovellusta varten ja miten sovellus käytännössä ottaen toimii. Tietokantarakenteet rakennettiin Microsoftin SQL Server Management Studio 17 sovelluksella ja demon kehitys tapahtui Microsoftin Visual Studio 2017 kehitysympäristössä ja tässä ohjelmointikielenä toimi c#. Sovellusta testattiin Postman-ohjelmalla, jolla pystyy lähettämään HTTP-kutsuja sovellukselle, ilman, että tarvitsee kirjoittaa ylimääräistä koodia.

Integraatio SaMi-järjestelmään ei loppujen lopuksi tämän työn aikana toteutunut ja sen toteuttaminen jäi odottamaan jatkokehitystä. Kyseisen integraation toteuttaminen olisi tuonut huomattavasti lisää laajuutta tähän työhön ja tämän takia päädyttiin toteuttamaan standardin ominaisuuksia toteuttava sovellus käyttäen, sille itse laadittuja tietokantarakenteita.

Avainsanat
SensorThings API, .Net, Internet of Things

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Joona Myllys			
Title of Thesis Integration of a Sensory framework to Savonia's Savonia Measurements System			
Date	13 May 2018	Pages/Appendices	26/0
Supervisor(s) Mr Mikko Pääkkönen, RDI specialist, Mr Keijo Kuosmanen, Senior Lecturer			
Client Organisation /Partners Mikko Pääkkönen, Savonia University of Applied Sciences			

Abstract

The purpose of this thesis was to get to know one sensory framework and to create a demo application implementing that framework. At the start there were two framework options and one was chosen as a more sensible option, due to it being more modern and supporting a larger number of different data formats.

The thesis examined the type of database structure that was built for the demo and how the demo works in practice. The database structures were built with Microsoft's SQL Server Management Studio 17 application and the development of the demo was done with Microsoft's Visual Studio 2017 development environment and c# was used as the programming language. The demo was tested using an application called Postman, which lets the user send HTTP requests to the demo without the need for additional code.

As the result of this thesis, a demo application was created but, the integration to the Savonia Measurements system was not done as it would have made the project so much broader and that is why the demo application was created with its own database to implement the standard features of the used framework. The integration remains to be implemented in later development phase.

Keywords
SensorThings API, .Net, Internet of Things

SISÄLTÖ

1 JOHDANTO 6
2 INTERNET OF THINGS 7
3 SENSOR OBSERVATION SERVICE (SOS) 8
4 SENSORTHINGS API 9
4.1 Entiteetit 10
4.2 Esimerkki 12
5 SAMI-JÄRJESTELMÄ 13
6 KÄYTETYT OHJELMISTOT JA TEKNIIKAT 14
7 INTEGRAATIO SAMI-JÄRJESTELMÄÄN 15
7.1 ThingTable 15
7.2 LocationTable 15
7.3 HistoricalLocationsTable 16
7.4 DatastreamTable 16
7.5 ObservedPropertiesTable 17
7.6 SensorTable 17
7.7 ObservationsTable 17
7.8 FeatureOfInterestTable 18
7.9 Lopullinen rakenne 18
8 ESIMERKKISOVELLUS 20
8.1 GET 20
8.2 POST 21
8.3 PATCH 23
8.4 DELETE 23
8.5 Jatkokehitys 24
9 YHTEENVETO 25
LÄHTEET JA TUOTETUT AINEISTOT 26

LYHENTEET JA MÄÄRITELMÄT

.NET: Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota voidaan käyttää tekemään kehitystyöstä nopeampaa, helpompaa ja tuottavampaa.

API: (Application Programming Interface) eli tarkoittaa ohjelmointirajapintaa, joka mahdollistaa eri ohjelmien välisen keskustelun.

ASP.NET: on Microsoftin kehittämä web-ohjelmistokehys. Se antaa kehittäjille eväät web-sivujen, web-ohjelmien ja web-palvelujen rakentamiseen.

CLIENT: Sovellus joka käyttää serverin tarjoamia palveluita.

HTTP: (Hypertext Transfer Protocol) on protokolla, jota WWW-palvelimet ja selaimet käyttävät tiedonsiirtoon.

IoT: (Internet of Things) Esineiden Internet.

ISO: (International Organization for Standardization) on kansainvälisesti toimiva organisaatio, joka tuottaa ja ylläpitää kansainvälisiä standardeja.

JSON: (JavaScript Object Notation) on yksinkertainen standardoitu tiedostomuoto tiedonvälitykseen.

O&M: (Observations and Measurements) on kansainvälinen standardi, joka määrittelee käsitteellisen skeeman havainnoille.

OASIS: on voittoa tavoittelematon järjestö, joka tukee avoimien standardien kehitystä ja käyttöönottoa kansainvälisen tietoyhteiskunnan saralla.

OGC: (Open Geospatial Consortium) on voittoa tavoittelematon järjestö, joka luo ja ylläpitää ilmaisia paikkatietoa koskevia standardeja.

OWS: (OGC Web Service Common) Web-palveluita määrittelevä standardi.

Palvelukeskeinen arkkitehtuurimalli: on malli, jossa eri toiminnot ja prosessit ovat suunniteltu toimimaan itsenäisinä palveluina.

REST: (REpresentational State Transfer) on HTTP-protokollaan perustuva arkkitehtuurimalli, jota käytetään monissa erilaisissa ohjelmointirajapinnoissa tiedonsiirtoa helpottamaan.

Resurssipohjainen arkkitehtuurimalli: on malli jota esimerkiksi REST toteuttaa.

SensorML: on OGC:n standardi, jolla kuvataan sensorien ja mittausten prosesseja.

SERVER(Palvelin): Palvelin tarkoittaa tietokonetta tai ohjelmistoa tietokoneella, jota voidaan kutsua palveluna tietoliikenteen yhteyden kautta.

SOAP: (Simple Object Access Protocol) on tietoliikenneprotokolla, joka pohjautuu XML-kieleen.

SQL: (Structured Query Language) on kyselykieli, jonka on kehittänyt IBM, sitä käytetään erilaisten hakujen, muutosten ja lisäysten tekemiseksi relaatiotietokantoihin.

SWE: (Sensor Web Enablement) on OGC:n standardikirjasto joka sisältää esimerkiksi myöhemmin mainittavat Sensor Observation Servicen ja SensorThings API:n.

UML: (Unified Modeling Language) on Object Management Groupin vuonna 1997 standardoima graafinen mallinnuskieli.

XML: (eXtensible Markup Language) on metakieli, jolla kuvataan tiedon rakennetta.

1 JOHDANTO

Opinnäytetyön tavoitteena on tutkia, onnistuuko sensorirajapinnan integrointi Savonian SaMi-tietojärjestelmään ja mitä muutoksia se mahdollisesti vaatisi tietojärjestelmään. Tämä siksi, että saataisiin mahdollisesti joku standardin mukainen rajapinta toteutettua SaMiin, joka helpottaisi tulevaisuudessa ylläpitoa ja uusien käyttäjien mukaan tuloa. Ideana oli myös tehdä ns. demosovellus toteuttaen sensorirajapinnan vaatimuksia.

Alkuperäinen suunnitelma oli lähteä tutkimaan SOS-rajapintaa (Sensor Observation Service), mutta muiden vaihtoehtojen noustessa esille, oli lähdettävä punnitsemaan mikä on järkevin rajapinta tämän opinnäytetyön kannalta, ottaen myös huomioon mahdollisen tulevaisuuden jatkokehityksen ja sovelluksien käytön Savonialla.

Lopulta rajapinnaksi valikoitui OGC SensorThings API. Tämä vaikutti monin tavoin paljon houkuttelevammalta vaihtoehdolta, ollessaan modernimpi (vasta vuonna 2016 julkaistu) ja käyttäessään REST-arkkitehtuurimallia. Se, että SensorThings käyttää RESTiä tarkoittaa myös sitä, että se on myös huomattavasti ylläpidettävämpi verrattuna SOS-rajapintaan, joka käyttää vanhempaa SOAP-tietoliikenneprotokollaa. RESTin helpompi ylläpidettävyys selittyy sen yksinkertaisuudella ja monipuolisuudella, REST esimerkiksi tukee useampia data formaatteja, SOAPin tukiessa pelkästään XML:ää.

Demoa kehittäessä tärkeäksi tuli koko ajan olla kartalla, siitä mitä vaatimuksia standardikuvaus sovellukselle määrittää. On esimerkiksi erittäin tärkeää tietää nojaako standardin toteutus johonkin toiseen rajapintaan tai muihin jo olemassaoleviin standardeihin, kuten tämä kyseinen Sensorthings nojaa vahvasti OData standardiin (OGC, 2016).

Tässä opinnäytetyössä toteutettiin perustason sovellus, joka osaa lukea dataa sisään, päivittää ja poistaa, sekä hakea dataa tietokannasta, joka myös toteutettiin standardin mukaisesti. Työssä kuvataan, miten toteutettu sovellus toimii ja kuvataan tarkasti, minkälainen tietokantarakenne on toteutettu.

2 INTERNET OF THINGS

Internet of Things (IoT) eli suomeksi sanoen esineiden internet on käsite, jolla tarkoitetaan sitä, kuinka jokapäiväiset esineet voivat olla yhteydessä internettiin ja toisiinsa, lähettäen dataa valmistajalle tai jollekin muulle osapuolelle. Internetiin kytkettävä esine voi olla niin yksinkertainen, kuin lämpömittari joka mittaa huoneiston lämpötilaa. Yleisin nyt jo tunnettu sovellus IoT:stä on etäluettava sähkömittari, joka lähettää kulutustiedot suoraan sähköyhtiölle (Logistiikan Maailma).

Yksi hyvä esimerkki IoT-tekniikan soveltamisesta on antureiden sijoittaminen jäteastioihin. Anturit mittaavat astian täyttöastetta ja kertovat järjestelmälle, milloin jäteastia tulisi tyhjentää. Tällä säästetään huomattavasti aikaa ja resursseja, kun voidaan suunnitella reitit tehokkaammin, koska ei tarvitse vajaita jäteastioita käydä tyhjentämässä jatkuvasti. (Logistiikan Maailma.)

Yksi suurimmista kuopista polullaan IoT:lla kuitenkin on tietoturva, kuka vastaa, jos ulkopuolinen taho kaappaa talon jääkaapin järjestelmän rikkoen sen. Tästä hyvä esimerkki on kuinka, helmikuussa 2018 itävaltalainen tietoturvayritys SEC Consult varoitti itkuhälyttimista, jotka ovat kaapattavissa yksinkertaisin keinoin. Hyökkääjä pystyisi esimerkiksi kuuntelemaan hälyttimiä tai puhumaan hälyttimen kautta, jopa videokuvan vakoileminen on mahdollista. Itkuhälyttimien valmistaja ei ole vastannut kyselyihin ja SEC:n yhteydenotot kiinalaisiin tietoturvaviranomaisiin olivat turhia, heidän mielestään haavoittuvuus laitteissa ei ollut julkaisun arvoinen. Tämä osaltaan on hyvä esimerkki siitä, mikä tekee haasteellista näiden ongelmien ratkaisussa. Toiset ihmisryhmät ja organisaatiot voivat nimittäin olla vahvastikin eri mieltä ongelmien laadusta ja kriittisyydestä. (SECLISTS, 2018.)

Tietoturvayhtiö F-securen tutkimusjohtaja Mikko Hyppönen on tuonut paljon tietoisuutta IoT:n riskeistä median kautta tavallisten kansalaisten tietoon. Hän on muun muassa sanonut haastattelussa, ettei laitevalmistajilla ole monessa tapauksessa mitään kiinnostusta panostaa tietoturvaan, onhan näiden yhtiöiden päätarkoitus tehdä rahaa, eikä parantaa kuluttajan asemaa. Hänen mielestään myös analytiikka on laitteiden valmistajille erittäin arvokasta esimerkiksi tarkkaillakseen laitteiden hajoamistiheyttä, tämä taas johtaa väistämättä siihen, että esineiden internetin vallankumous tulee tapahtumaan. Riskinä on, että kun yrityksillä on kiire markkinoille, jää tietoturva toissijaiseksi asiaksi ja tämä voi taas johtaa siihen, että älyjääkaappisi pystyy huolehtimaan ostoslistoistasi, mutta samalla antaa verkkorikolliselle keinon päästä tyhjentämään pankkitilisi. (F-secure, 2018.)

Kaiken kaikkiaan IoT on tulevaisuuden asia, kunhan vain tietoturva ja muut ongelmat saadaan selvitettyä, tulee se avaamaan meille vielä monia ovia uusiin elämää helpottaviin ja tehostaviin ratkaisuihin. Tulevaisuudessa voi olla hyvinkin mahdollista, että pystymme kotona työaamuna laskemaan järkevimmän reitin töihin reaaliajassa, sen perusteella, minkälainen on ruuhkatilanne ja laittamaan saunan päälle jo kotimatkalla.

3 SENSOR OBSERVATION SERVICE (SOS)

SOS on standardi ja kuuluu OGC:n perheeseen, joka kokonaisuudessaan muodostaa OGC SWE Frameworkin. SOS:in funktionaalisuus SWE:n sisällä on tuottaa standardoitu pääsy käsiksi mitattuihin sensorihavantoihin, kuin myös kyseisten sensoreiden kuvauksiin. Havaintojen koodaukseen(Encoding) käytetään O&M-standardia. Sensorien kuvauksien koodaukseen(Encoding) käytetään Sensor Model Languagea (SensorML). (OGC, 2012, 10.)

Standardi käyttää SOAP-tietoliikenneprotokollaa kaikkiin toimintoihinsa ja tämän lisäksi KVP-sidosta(Key-Value-Pair) ydinoperaatiohin, kuten tuloksien käsittelyyn liittyvissä toimenpiteissä. Tulevissa versioissa tai lisäosissa tähän standardiin voidaan mahdollisesti, tulla lisäämään RESTful tyyppiset palvelut. (OGC, 2012, 13.)

SOS käyttää tiedostomuotona XML:ää ja se on arkkitehtuuriltaan palveluorientoitunut. Uusien sensorien luomiseksi SOS:ssa tulee käyttää sille spesifisiä rajapintoja, kuten RegisterSensor() tai InsertObservation(). Kyseessä olevalla standardilla ei ole minkäänlaista tukea olemassaolevien sensorien tietojen päivittämiselle ja esimerkiksi tilanteessa, jossa standardin läpi on ladattu virheellisiä havaintotietoja, ei siinä ole tukea niiden muokkaamiselle tai poistamiselle, vaan käytännössä pitäisi aina pyytää järjestelmän ylläpitäjää poistamaan tai muokkaamaan käsin tietokannasta nämä tiedot. Sillä ei ole myöskään tukea minkäänlaiselle linkitetylle datalle, eikä asiakasohjelmien kautta pysty tekemään muokattuja hakuja, joilla pystyisi esimerkiksi parametreillä vaikuttamaan siihen, mitä tietoa kannasta haetaan. Myöskään useamman eri entiteetin haku yhdellä kutsulla ei ole tuettu ominaisuus SOS:ssa. (Sensorup, 2016.)

```
<?xml version="1.0" encoding="UTF-8"?>
 <soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope" xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope htt
http://www.opengis.net/sos/2.0 http://schemas.opengis.net/sos/2.0/sos.xsd" xmlns:sos="http://www.opengis.net/sos/2.0" xmlns:wsa="http://
 <soap12:Header>
 <wsa:To>http://www.ogc.org/SOS </wsa:To>
 <wsa:Action>http://www.opengis.net/def/serviceOperation/sos/obsInsertion/2.0/InsertObservation </wsa:Action>
 <wsa:ReplyTo>
 <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous </wsa:Address>
 </wsa:ReplyTo>
 <wsa:MessageID>http://my.client.com/uid/msg-0010 </wsa:MessageID>
 </soap12:Header>
 <soap12:Body>
 <sos:InsertObservation service="SOS" version="2.0.0">
 <sos:offering>http://www.my_namespace.org/water_gage_2_observations </sos:offering>
 <sos:observation>
 <om:OM_Observation gml:id="obsTest1">
 <om:type
 xlink:href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement"/>
 <om:phenomenonTime>
 <gml:TimeInstant gml:id="phenomenonTime">
 <gml:timePosition>2010-08-31T17:45:15.000+00:00 </gml:timePosition>
 </gml:TimeInstant>
 </om:phenomenonTime>
 <om:resultTime xlink:href="#phenomenonTime"/>
 <om:procedure xlink:href="http://www.my_namespace.org/sensors/Water_Gage_2"/>
 <om:parameter>
 <om:NamedValue>
 <om:name xlink:href="http://www.opengis.net/req/omxml/2.0/data/samplingGeometry"/>
 <om:value>
 <gml:Point gml:id="SamplingPoint2">
 <gml:pos srsName="http://www.opengis.net/def/crs/EPSG/0/4326">54.9 10.52 </gml:pos>
 </gml:Point>
 </om:value>
 </om:NamedValue>
 </om:parameter>
 <om:observedProperty xlink:href="http://sweet.jpl.nasa.gov/2.0/hydroSurface.owl#WaterHeight"/>
 <om:featureOfInterest xlink:href="http://wfs.example.org?request=getFeature&amp;amp;amp;amp;featureid=river1"/>
 <om:result xsi:type="gml:MeasureType" uom="urn:ogc:def:uom:OGC:m">0.28 </om:result>
 </om:OM_Observation>
 </sos:observation>
 </sos:InsertObservation>
 </soap12:Body>
 </soap12:Envelope>
```

KUVA 1 SOS-standardin mukainen uuden havainnon lisäämiskutsu. (Sensorup, 2016)

4 SENSORTHINGS API

SensorThings API on OGC:n standardi joka tarjoaa avoimen ja yhtenäisen rajapinnan, jolla yhdistää IoT-havaintolaitteita, dataa ja sovelluksia verkon ylitse. SensorThings API koostuu kahdesta osasta, jotka ovat Part 1 – Sensing ja Part 2 – Tasking. Sensing osuus tarjoaa standardoidun tavan hallita ja hakea havaintoja ja metadataa heterogeenisistä IoT Sensorijärjestelmistä. Tasking osuus on tällä hetkellä vasta kehittelyssä ja tässä opinnäytetyössä pureudutaankin vain Sensing osuuteen. (OGC, 2016.)

SensorThings API:n tarkoituksena on yksinkertaistaa ja nopeuttaa IoT-sovellusten kehitystä ja tämän lisäksi, muuntaa lukuisat irtonaiset IoT-järjestelmät yhdeksi yhdistyneeksi alustaksi, jolla erilaisia toimia pystytään toteuttamaan. Tätä avointa standardia voivat kehittäjät käyttää yhdistääkseen erilaisia IoT-laitteita ja luodakseen innovatiivisia sovelluksia, ilman, että tarvitsee huolia eri laitteiden ja palveluiden erilaisista käytännöistä koko ajan. Sensorthingsiä pystytään myös soveltamaan laitevalmistajien toimesta siten, että heidän laitteensa ovat jo tehtaalta tullessa sellaisia, että ne pystyvät ottamaan yhteyttä OGC:n standardin mukaisiin palvelimiin ympäri maailmaa. (OGC, 2016.)

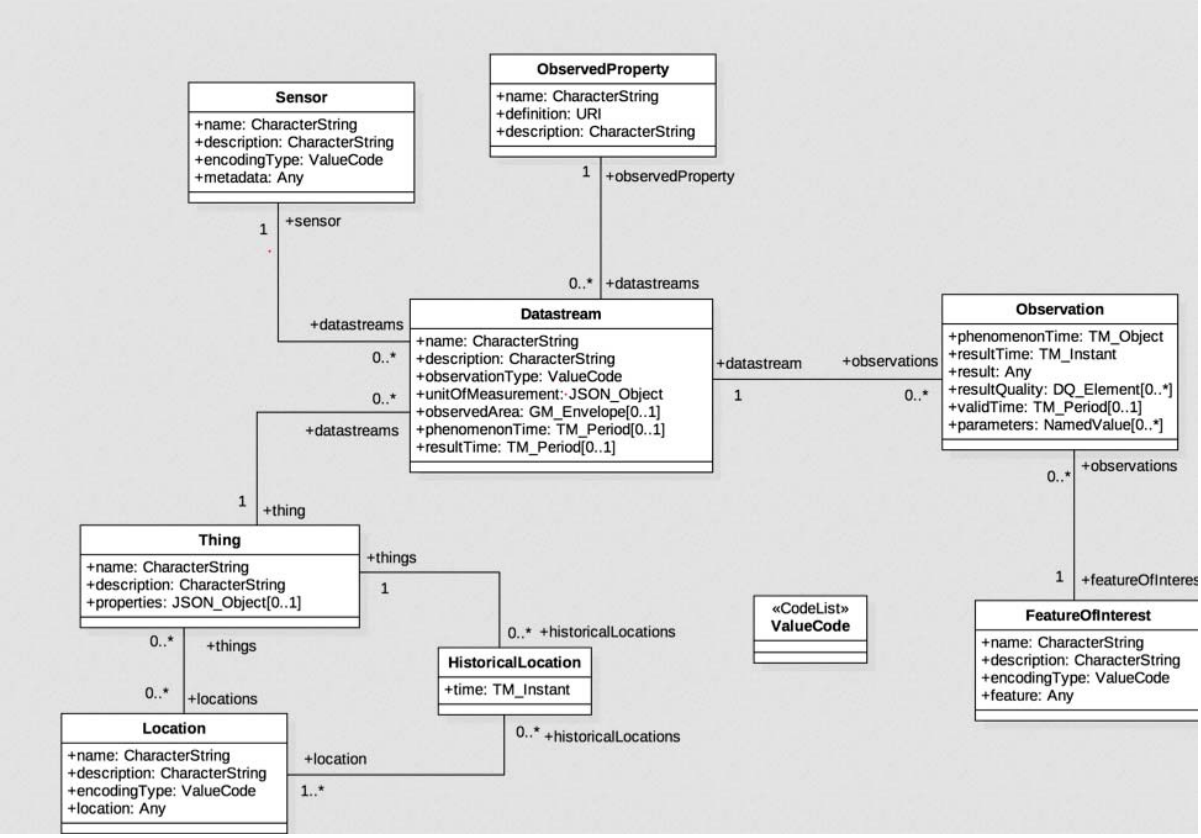
Part 1 – Sensing, tarjoaa IoT-laitteille ja sovelluksille mahdollisuuden luoda, lukea, päivittää ja poistaa (HTTP:n POST, GET, PATCH, DELETE), IoT dataa ja metadataa SensorThings palvelun kautta. Se on suunniteltu O&M standardin mallin mukaisesti. Mallin perusidea on, että jokainen havainto (Observation) on tapahtuma joka tuottaa tuloksen, jonka arvo on arvio tutkittavan kohteen ominaisuudesta (esim. tuulen nopeus). Havainto-tapahtuma luokitellaan sen syntyajan, havainnossa mitattavan tapahtuman ja mittaavan sensorin/anturin mukaan. Tämän lisäksi SensorThings API:ssa kuvataan keskeisinä objekteinä asioita eli Things, joka määritellään Telecommunication Standardization Sectorin mukaan näin: ”an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks” (ITU-T Y.2060), eli vapaasti suomentaen, ”Thing on fyysisen tai virtuaalisen maailman objekti, joka voidaan todeta ja integroida osaksi yhteysverkkoa”. Thing sanalle ei ole suomen kielessä vakiintunutta termiä. (OGC, 2016.)

SensorThings API käyttää JSON-tiedostomuotoa ja on arkkitehtuuriltaan resurssiorientoitunut. Verraten SOS:iin on SensorThingsillä huomattavasti yksinkertaisemmat kutsut ja se myös tarjoaa paljon enemmän joustavuutta käyttäjälle. Ollessaan RESTful palvelu, tapahtuu uusien sensorien ja havaintojen lisäykset HTTP:n POST-kutsulla, Poistaminen DELETE-kutsulla ja sensorien ja havaintojen päivittäminen PATCH-kutsulla. Se, myös poiketen SOS:sta, tarjoaa tuen linkitetylle datalle JSON-LD metodilla, tämän lisäksi se tarjoaa $select parametrilla tuen sille, että haetaan asiakasohjelman antamien parametrien mukaan, vain ne kentät joita pyydetään. Viimeisenä listattuna etuna SOS:iin verrattuna on $expand parametri, jolla voi hakea useampia entiteettejä yhdellä kutsulla. (Sensorup, 2016.)

SensorThings API seuraa ODatan määrityksiä entiteettien pyynnöille, joka tarkoittaa sitä, että entiteetin ohjausinformaatio, resurssien polkujen käytöt, kyselyvaihtoehdot, oleelliset JSON-koodaukset ja eräprosessointi pyynnöt, seuraavat Odata 4.0: aa. Tällä on tähdätty siihen, että kehittäjät, joille OData on jo tuttua, voivat kehittää SensorThings sovelluksia helposti. On kuitenkin huomattava, ettei SensorThings seuraa ODatan yleistä järjestelmänkuvauskieltä ja tästä johtuen ei tue sen metadatapalvelu-entiteettimallia. SensorThingsiä ei voi tästä syystä katsoa OData yhteensopivaksi API:ksi. SensorThingsin tulevaisuuden kehityksessä pyritään yhdenmukaistamaan näiden kahden välisiä käytäntöjä. (OGC, 2016.)

4.1 Entiteetit

SensorThings API Part 1 – Sensing, koostuu 8 eri entiteetistä (Kuva 2), joista jokainen pystyy Luomaan, Lukemaa, Päivittämään ja Poistamaan aiemmin mainituin HTTP: verbein.



KUVA 2 UML-kaavio SensorThings API:n entiteeteistä ja niiden relaatioista. (OGC.)

Thing on fyysisen tai virtuaalisen maailman objekti joka voidaan todeta ja integroida osaksi yhteysverkkoa. Thing voi esimerkiksi olla henkilö, jonka painoa halutaan seurata. Kuva 2:n kaavion perusteella pystymme päättelemään, että Thing on objekti, jolla on nolla tai useampia Locationia (sijaintia kuvaava entiteetti), nolla tai useampia Datastreamia ja nolla tai useampia HistoricalLocationeita (kuvaa entisiä sijainteja).

Location entiteetti kuvaa Thingin sijaintia. Entiteetin sisällä sijainnin kuvaus riippuu sen koodaustyypistä, se voi olla kuvattu koordinaateilla erittäin tarkasti, tai sitten sijainti voi olla joku suuripiirteisempi, kuten ”Savonian Opistotien kampus”. Locationilla voi olla nolla tai useampia Thingejä ja nolla tai useampia HistocialLocationeita.

HistoricalLocation kertoo Thingin viimeisen tiedetyn Locationin aikaleiman ja tämän lisäksi aikaleimat edellisistä Locationeista. Tämä tieto tulee tarpeelliseksi esimerkiksi liikkuvissa kohteissa, joissa on sensoreita, kuten nykypäivänä dronet. Yhdellä HistoricalLocationilla voi olla yksi tai useampia Locationeita, mutta vain yksi Thing.

Datastream ryhmittelee joukon Observationeita, jotka mittaavat samaa ObservedPropertyä ja jotka tuotetaan samaa Sensoria käyttäen. Datastream onkin SensorThings API:ssa keskeinen entiteetti, kuten kuva 2:n kaavio osoittaa. Yhdellä Datastreamilla on vain yksi Thing, Yksi Sensori, Yksi ObservedProperty ja nolla tai enemmän Observationeita (havaintoja).

Sensor on väline, joka havainnoi ominaisuutta tai ilmiötä tavoitteenaan tuottaa joku arvo tälle tapahtumalle. Sensor tuottaa dataa Datastreamille. Esimerkkeinä Sensoreista voisi mainita lämpömittarit, tutkat ja jopa tuuliviiri lasketaan Sensoriksi, mittaahan se tuulen suuntaa. SensorThings API:ssa Sensorilla on suhde vain Datastreamin kanssa. Datastremejä joihin Sensori voi tuottaa Observationeita voi olla nolla tai useampia.

ObservedPropertyllä kuvataan ilmiötä, jota kyseinen Datastream tutkii, eli esimerkiksi tuulen nopeus tai alueen lämpötila. ObservedPropertyn relaatiot on myös Sensorin tapaan vain Datastreamiin. Kaikki datastreamin Observationit tutkivat samaa ObservedPropertyä, mutta myös muissa Datastreameissä voi olla Observationeita, jotka tutkivat samaa ObservedPropertyä, eli ObservedPropertyllä voi olla suhde moneen eri Datastreamiin.

Observation on yksittäinen havainto. Observation on yksi ominaisuuden, eli FeatureOfInterestin arvo tai määrite, jonka Sensori mittaa. Observationin relaatiot ovat Datastreamiin ja FeatureOfInterestiin. Observationit ovat arvoja aina yhdessä Datastreamissä ja Observationit tulevat aina yhdestä FeatureOfInterestistä. Observationin olemassaolo tarkoittaa sitä, että jostain ilmiöstä on määritetty arvo. FeatureOfInterest kuvaa tätä ilmiötä tai oliota, josta on saatu nyt arvo. FeatureOfInterest voi esimerkiksi olla rakennus, jonka sähkönkulutusta kuvataan, tällöin tälle rakennukselle ollaan mittauksessa saatu joku arvo. Observationia sisään lukiessa sille tulee aina määrittää joku FeatureOfInterest. Jos Observationin luonnissa tälläistä ei löydy, järjestelmä generoi uuteen Observationiin liittyvästä Datastreamista, sen Thing-entiteetin Locationin mukaisen FeatureOfInterestin, eli toisinsanoen Thingin Locationin tiedot kopioidaan FeatureOfInterestiksi.

4.2 Esimerkki

Halutaan tutkia, kuinka monta lainattavaa polkupyörää on vapaana tällä hetkellä Savonian Opistotien kampuksella. Datastream kuvaa sitä, kuinka monta niitä pyöriä on kyseisellä hetkellä vapaana. Thing on Opistotien Kampuksen paikka, josta näitä pyöriä pystyy lainaamaan, olkoon se tässä tapauksessa vaikka ”Opistotien kirjasto”. Thingillä on location, joka tässä tilanteessa olisi koordinaatit tähän ”Opistotien kirjasto” paikkaan. HistoricalLocationeita ei olisi, koska kirjastot harvemmin vaihtavat paikkaa. Sensorina tässä tilanteessa toimii laite, joka pitää kirjaa siitä, kuinka monta pyörää on kullakin hetkellä saatavilla. ObservedProperty on lukema pyöristä, jotka ovat kirjastossa vapaana lainattaviksi. Observation on tietenkin se lukumäärä jonka tämä sensori mittaa, esimerkiksi Observation yhdeltä mittauskerralta voisi olla ”5”. Observationilla on FeatureOfInterest joka tässä tapauksessa olisi kuvaava tieto siitä, että missä sijainnissa on tämä mittaus tehty. Tämä on yksinkertainen esimerkki SensorThings API:n mukaisesta kokonaisesta objektista.

5 SAMI-JÄRJESTELMÄ

SaMi-tietojärjestelmä (Savonia Mittaukset) on Savonia Ammattikorkeakoulun avoimen lähdekoodin järjestelmä, joka on kehitetty yhtenäistämään eri projekteissa tarvittavia mittatietokantoja. Monessa projektissa on tarve tallentaa erilaista tietoa ja aiemmin jokaiselle projektille oli luotava oma taustajärjestelmä ja tämä johti siihen, että projektien tiedot olivat hajallaan eri paikoissa. SaMi-järjestelmä poistaa tarpeen luoda taustajärjestelmä jokaiselle projektille. Projektien tiedot löytyvät järjestelmän kautta keskitetysti ja niitä voidaan uudelleen soveltaa, nopeuttaen uusien järjestelmien kehittämistä. (Pääkkönen, 2018)

Järjestelmää voidaan käyttää myös opetuksen tukena, tarjoten esimerkkejä opiskelijoille oikeassa elämässä toimivien tietokantojen, käyttöliittymien ja muiden tietojärjestelmien komponenttien toiminnasta. Järjestelmää onkin jo käytetty esimerkiksi Kuopion Veden ja Niiralan Kulma Oy:n projekteissa, liittyen vesijärjestelmien seurantaan. (Savonia)

Itse järjestelmä on Savoniassa kehittynyt useiden tuotekehityshankkeiden, opiskelijaprojektien ja tietohallinnon hallinnoimien projektien tuloksena. Järjestelmä on julkaistu MIT-lisenssin alaisena Github-palvelussa. Suomalainen ohjelmistoyritys Keypro Oy käyttää järjestelmää hyväksi omassa verkkotietojärjestelmässään, jonka tarkoituksena on hallita vesijohto- ja viemäriverkkoa. (Savonia)

6 KÄYTETYT OHJELMISTOT JA TEKNIIKAT

Työ on .NET-ympäristössä toteutettu sovellus, jonka ohjelmistokehitysympäristönä toimi Microsoftin Visual Studio. Visual Studio on microsoftin kehittämä ohjelmankehitysympäristö, jolla voi tehdä monia erilaisia sovelluksia, kuten verkkosivuja, webohjelmia tai mobiilisovelluksia. Se tukee useita eri kieliä joita ovat esimerkiksi JavaScript, C++, C# ja Python. Siinä on myös sisäänrakennettu tuki yli 4000 lisäosalle. Visual Studiosta on tällä hetkellä (2018 maaliskuu) saatavilla ilmainen Community versio Visual Studion sivustolta. (Microsoft, 2018)

Ohjelmointikielenä toimi C# joka on moderni ja yksinkertainen. Se on olio-orientoitunut ja symbioottinen .NET alustan kanssa. Syntaksiltaan ja ohjausrakenteiltaan se muistuttaa C:tä ja C++: aa. Ideana kielen kehittämisessä olikin pitää hyvät puolet näistä kielistä ja muuttaa syntaksia vain, jos siihen on selkeä tarve. (Sivonen 2004)

REST-kutsujen testaamiseen käytettiin Postman-ohjelmaa, joka on APIen kehitysympäristö, jolla pystyy lähettämään HTTP-kutsuja ohjelmille kirjoittamatta yhtään erillistä koodia sitä varten. Tämä ohjelma oli erittäin hyödyllinen ja yksinkertaisti huomattavasti työn eri funktioiden testaamista.

Microsoft SQL Server Management Studio on nimensä mukaisesti Microsoftin kehittämä integroitu ympäristö minkä tahansa SQL-ympäristön hallinnointiin. SSMS tarjoaa mahdollisuuden tehdä kutsuja, mahdollisuuden suunnitella ja ylläpitää tietokantoja ja tietovarastoja, olivat ne sitten omalla tietokoneella tai pilvessä. Tätä ohjelmaa käytettiin rakentamaan tietokanta ja sen vaatimat taulut. (Microsoft)

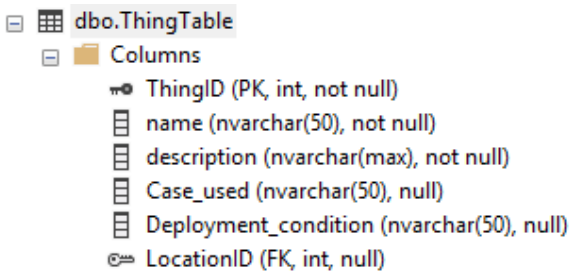
Työssä myös sovellettiin ODataa (Open Data Protocol), joka on OASIS:in standardi, se määrittelee joukon parhaita käytäntöjä RESTful API:en rakentamiseen (Odata, 2017). Odata auttaa kehittäjää yksinkertaistamalla kyselyitä ja datan jakamista sovellusten välillä. Odatalla pystyy hakemaan dataa monista erilaisista lähteistä, joita on esimerkiksi relaatiotietokannat, tiedostojärjestelmät ja perinteiset web-sivustot (OASIS, 2017).

7 INTEGRAATIO SAMI-JÄRJESTELMÄÄN

Järjestelmää varten on luotava oma tietokanta, johon tulee kahdeksan taulua, joista jokainen kuvaa yhtä entiteettiä Sensorthings API:ssa. Jokaisella entiteetillä on oma uniikki ID-kenttä, joka toimii PrimaryKey:nä kussakin taulussa. Taulut nimettiin tässä työssä selvyyden vuoksi entiteettien nimien perusteella, tämä myös tehtiin siksi, että demon koodin lukeminen ja hahmoittaminen olisi selkeämpää. Taulujen nimiksi tulivat: DatastreamTable, FeatureOfInterestTable, HistoricalLocationsTable, LocationsTable, ObservationsTable, ObservedPropertiesTable, SensorTable ja Thingtable. Seuraavissa osioissa jokaisesta taulusta tarkemmin.

7.1 ThingTable

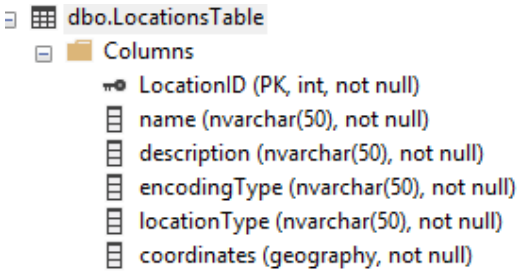
ThingTable sisältää kuusi saraketta, jotka ovat int-tyyppinen PrimaryKey ThingID, nvarchar-tyyppiset kentät name, description, Case_used ja Deployment_condition ja int-tyyppinen LocationID. Case_used ja Deployment_condition ovat kenttiä, joita voi tulla Thingiä luodessa "properties" objektissa sen sisällä, objektin sisältö on määritelty siten, että sen sisällä voi tulla mitä tahansa kenttiä, nämä kaksi olivat esimerkeissä olevat. ThingTablen LocationID:llä on ForeignKey-suhde LocationTablen LocationID:n kanssa. Pakollisia kenttiä ovat ThingID, joka automaattisesti generoituu, sekä name ja description.



KUVA 3 ThingTable (Joona Myllys, 2018)

7.2 LocationTable

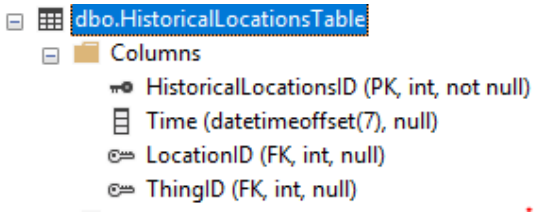
LocationTable sisältää kuusi saraketta: int-tyyppinen LocationID, joka toimii PrimaryKeynä, nvarchar-tyyppiset name, description, encodingType ja locationType ja viimeiseksi geography tyyppinen coordinates-sarake. Jokainen näistä kentistä on pakollinen.



KUVA 4 LocationTable (Joona Myllys, 2018)

7.3 HistoricalLocationsTable

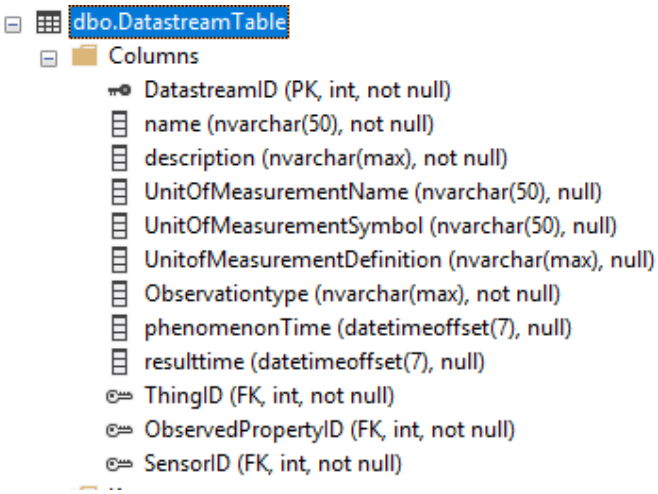
HistoricalLocationsTable sisältää vain neljä saraketta: int-tyyppiset HistoricalLocationsID, LocationID ja ThingID, sekä datetimeoffset-tyyppisen Time-sarakkeen. HistoricalLocationsID on PrimaryKey, LocationID ja ThingID ovat ForeignKeynä. HistoricalLocationsTablen LocationID viittaa LocationTablen LocationID:hen ja ThingID viittaa ThingTablen ThingID:hen.



KUVA 5 HistoricalLocationsTable (Joona Myllys, 2018)

7.4 DatastreamTable

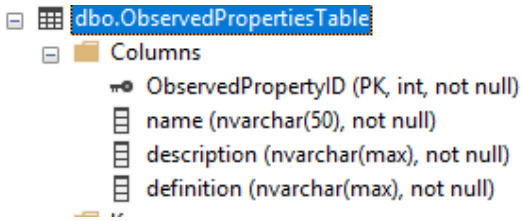
DatastreamTablessa on 12 saraketta, ollen näin isoin tauluista. Taulusta löytyy int-tyyppiset DatastreamID, ThingID, ObservedPropertyID ja SensorID. Seuraavaksi nvarchar-tyyppiset name, description, UnitOfMeasurementName, UnitOfMeasurementSymbol, UnitOfMeasurementDefinition ja ObservationType. Viimeisenä datetimeoffset-tyyppiset phenomenonTime ja resulttime. PrimaryKey on DatastreamID. ForeignKeyt ovat ThingId joka viittaa ThingTablen ThingID:hen, ObservedPropertyID viittaa ObservedPropertyTablen saman nimiseen kenttään ja SensorID SensorTablen SensorID:hen. Kentistä pakollisia ovat name, description, observationtype, ThingID, ObservedPropertyID ja SensorID.



KUVA 5 DatastreamTable (Joona Myllys, 2018)

7.5 ObservedPropertiesTable

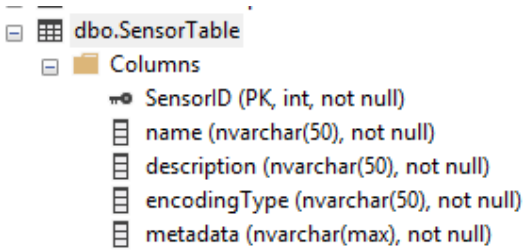
ObservedPropertiesTable koostuu neljästä sarakkeesta, nämä ovat int-tyyppinen ObservedPropertyID ja nvarchar-tyyppiset name, description ja definition. Taulussa kaikki kentät ovat pakollisia. PrimaryKey on ObservedPropertyID.



KUVA 6 ObservedPropertiesTable (Joona Myllys, 2018)

7.6 SensorTable

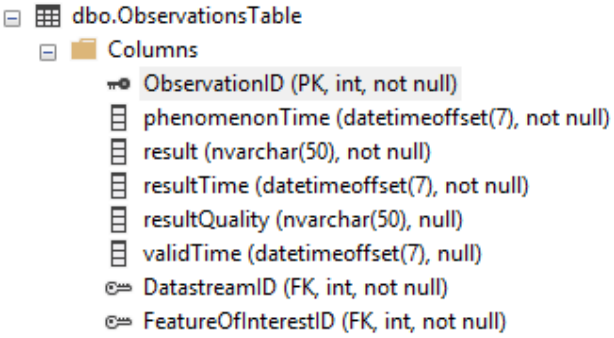
SensorTable sisältää viisi saraketta: SensorID, joka on int-tyyppinen ja neljä nvarchar-tyyppistä saraketta, jotka ovat: name, description, encodingType ja metadata. Kaikki kentät ovat pakollisia tässä taulussa. PrimaryKey on SensorID.



KUVA 7 SensorTable (Joona Myllys, 2018)

7.7 ObservationsTable

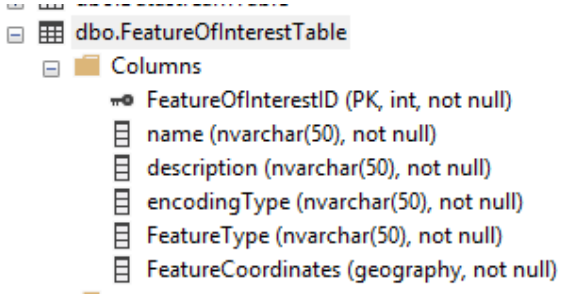
ObservationsTable koostuu kahdeksasta sarakkeesta. Tyypiltään int olevia sarakkeita on 3, ne ovat ObservationID, DatastreamID ja FeatureOfInterestID. Taulusta löytyy myös 3 saraketta jotka ovat tyypiltään datetimeoffset ja ne ovat phenomenonTime, resultTime ja validTime. Viimeiset kaksi saraketta ovat nvarchar-tyyppisiä: result ja resultQuality. Pakollisia kenttiä ovat ObservationID, phenomenonTIme, result, resultTime, DatastreamID ja FeatureOfInterestID. PrimaryKey on ObservationID ja ForeignKeyt ovat DatastreamID, joka viittaa DatastreamTablen DatastreamID:hen ja FeatureOfInterestID, joka viittaa FeatureOfInterestTablen FeatureOfInterestID:hen.



KUVA 8 ObservationsTable (Joona Myllys, 2018)

7.8 FeatureOfInterestTable

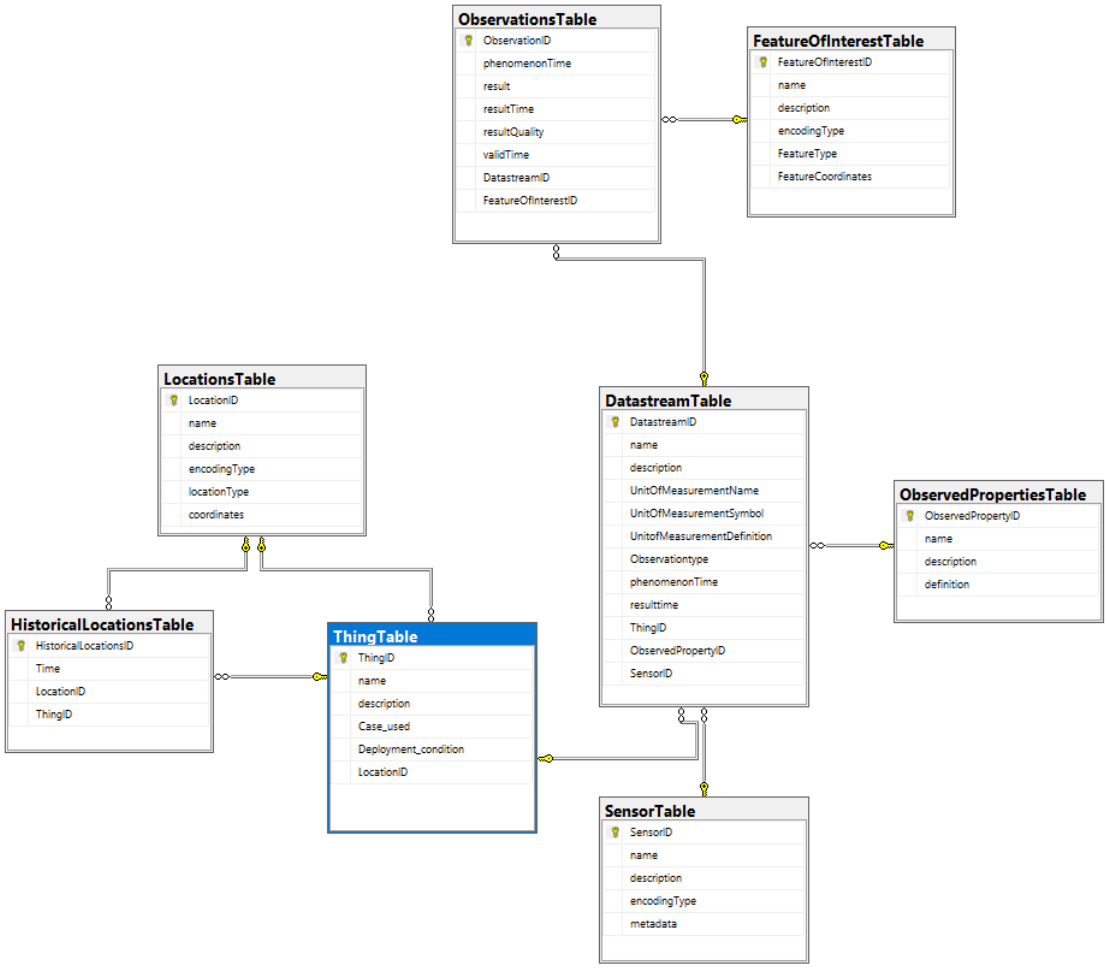
FeatureOfInterestTable koostuu kuudesta sarakkeesta, jotka ovat int-tyyppinen FeatureOfInterestID, joka samalla toimii PrimaryKeynä, neljä nvarchar-tyyppistä saraketta: name, description, encodingType ja FeatureType ja viimeisenä FeatureCoordinates joka on geography-tyyppinen. Kaikki kentät ovat pakollisia.



KUVA 9 FeatureOfInterestTable (Joona Myllys, 2018)

7.9 Lopullinen rakenne

Nämä kahdeksan taulua muodostavat tietokantarakenteen, jonka päälle demoa lähdettiin rakentamaan. Lopullinen rakenne ja taulujen suhteet näkyvät kuvasta 10, josta huomataan, että rakenne on käytännön toteutus kuvan 2 UML-kaaviosta.



KUVA 10 Lopullinen tietokantarakenne. (Joona Myllys, 2018)

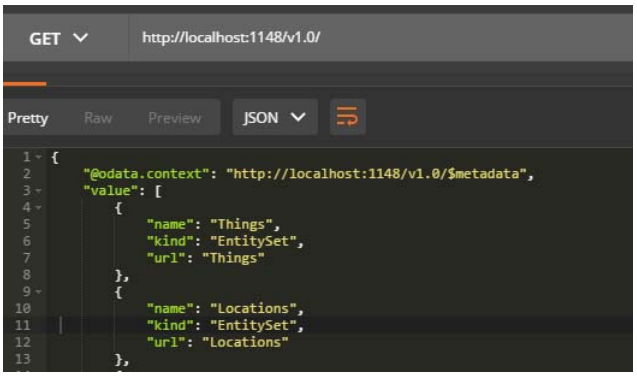
8 ESIMERKKISOVELLUS

Demosovellus on toteutettu .Net Framework 4.6.1:llä, käyttäen nykyisin API-nimellä kulkevaa templatea (entinen Web-API). Code First from database-lähestymistavalla luotiin edellisessä osiossa kuvattujen taulujen mukaiset luokat ja tietokantaan yhteys, jotta sovellus ja tietokanta voivat keskustella molemminsuuntaisesti.

Sovellus toimii siten, että HTTP-metodin ja URI:in (Uniform Resource Identifier) perusteella käyttäjän lähettämä kutsu osaa mennä oikeaan funktioon, joka ottaa vastaan parametreina mahdollisen, kyseisen kutsun body-lohkossa tulevan tiedon tai URI:n liitteenä olevan lisätiedon, riippuen HTTP-metodista, on mahdollista, että kumpikin näistä tiedoista välitetään funktiolle tai toisissa tapauksissa ei parametreja tarvita ollenkaan (esimerkiksi yksinkertainen GET-kutsu). Ohjelmassa tuettuja HTTP-metodeja on neljä kappaletta: GET, jolla haetaan tietokannasta dataa, POST, jolla luodaan uutta dataa tietokantaan, PATCH, jolla muokataan tietokannassa olevaa dataa ja DELETE, jolla poistetaan olemassaolevaa dataa. Demossa keskityttiin tekemään ihan perustason toteutusta ja täten ihan perinteiset kutsut vain toimivat, ODatan mukaiset monimutkaisemmat kutsut ovat myös mahdollista jatkokehityksessä toteuttaa sovellukseen, näitä on esimerkiksi datan tarkemmat haut tietokannasta ja liittyvien entiteettien hakeminen yhdellä kutsulla. Seuraavassa osiossa kuvataan tarkemmin, minkälaisia kutsuja ohjelmalle pystyy lähettämään.

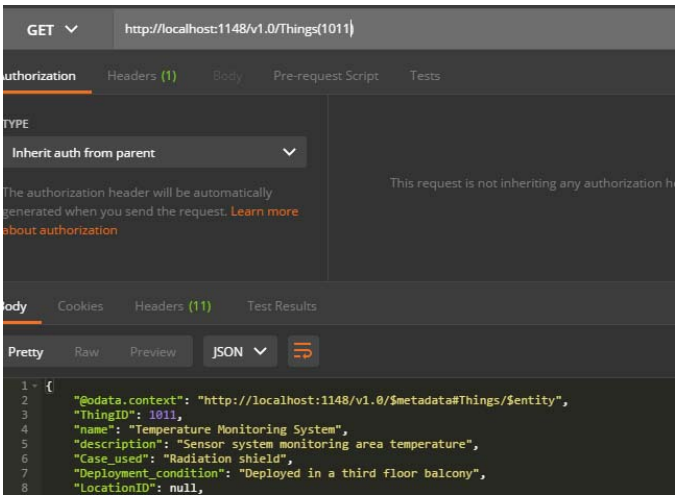
8.1 GET

Ohjelmalle lähettämällä GET-kutsun, ilman, että määrittelee mitään entiteettiä, palauttaa se listan kaikista mahdollisista entiteeteistä ja niiden tyypistä, sekä siitä nimestä millä niitä tulisi kutsua URI:ssa.



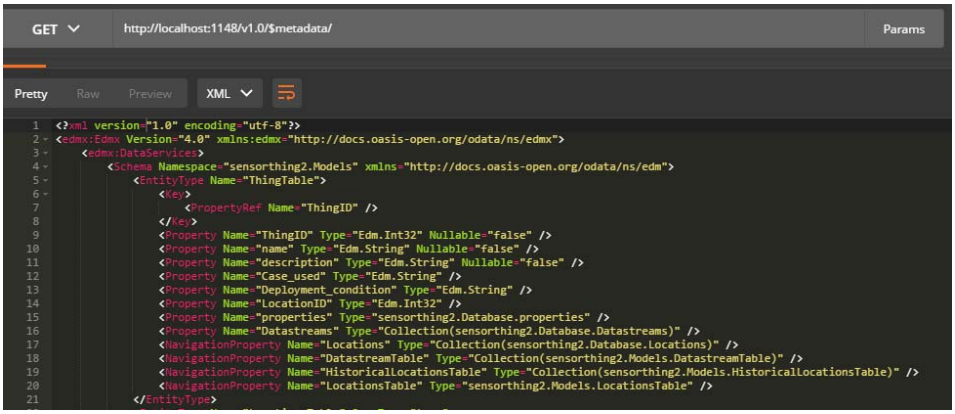
KUVA 11 Tyhjä GET-kutsu. (Joona Myllys, 2018)

GET-kutsulla, lisäämällä kuvassa 11 näkyvään URI:in entiteetin kutsun voidaan hakea kaikki kyseiset entiteetit tietokannasta ja jos halutaan hakea vain yksi haluttu objekti, voidaan perään laittaa sulkeissa sen objektin id, jota haetaan (kuva 12).



KUVA 12 Yhden objektin haku GET-kutsulla, palautuva data. (Joona Myllys, 2018)

Kolmantena vaihtoehtona GET-kutsuilla on antaa perus URI:n perään parametrina ”$metadata”, joka näyttää kaikkien ohjelmassa olevien tietorakenteiden tiedot (kuva 13). Tällä tiedolla käyttäjä voi selvittää niin sanotusti, menemättä konepellin alle, minkä tyyppistä dataa ja millä nimellä hänen tulisi ohjelmalle lähettää, jotta se toimisi oikein.

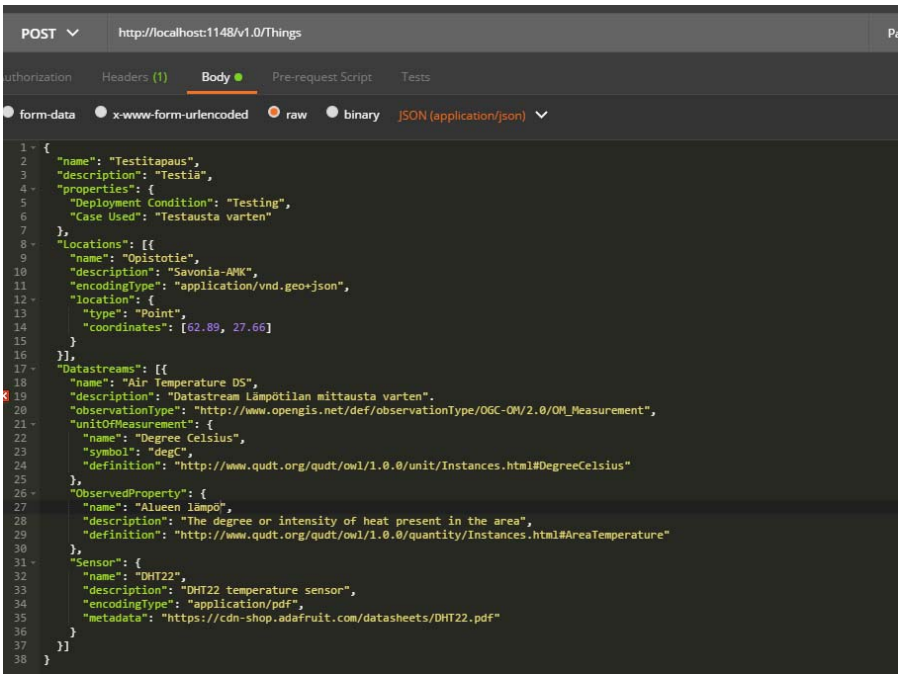


KUVA 13 GET-kutsulla metadatan haku. (Joona Myllys, 2018)

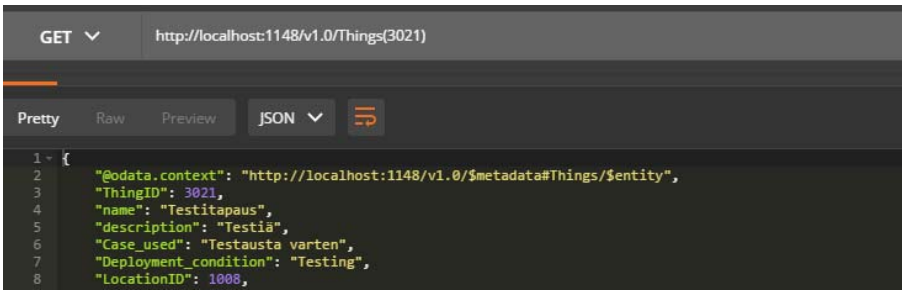
8.2 POST

HTTP-komento POSTia käytetään uuden datan lisäämiseen. Käyttäjä lähettää ohjelmalle kutsun, minkä objektin se haluaa luoda ja kutsun body-osassa se lähettää lisättävän datan. Sovelluksessa on mahdollista lisätä useampia objekteja eri entiteettien osalta samalla kutsulla, tämä toimii käytännössä vain Thing ja Datastream entiteettien kohdalla, johtuen siitä, että esimerkiksi Datastream objektilla on oltava viittaukset olemassa oleviin Sensor-, Thing- ja ObservedProperty-objekteihin, jos näitä ei ole olemassa, on ne luotava, jotta Datastream voidaan luoda. Kuvassa 14 kuvataan kuinka, samalla kutsulla luodaan Thing-, Location-, Datastream-, ObservedProperty- ja Sensor-objektit tietokantaan. Kuvissa 15 ja 16 todennetaan juuri luodut objektit tietokannasta. Onnistunut POST-komento palauttaa luodun objektin. Standardin mukaisesti pitäisi

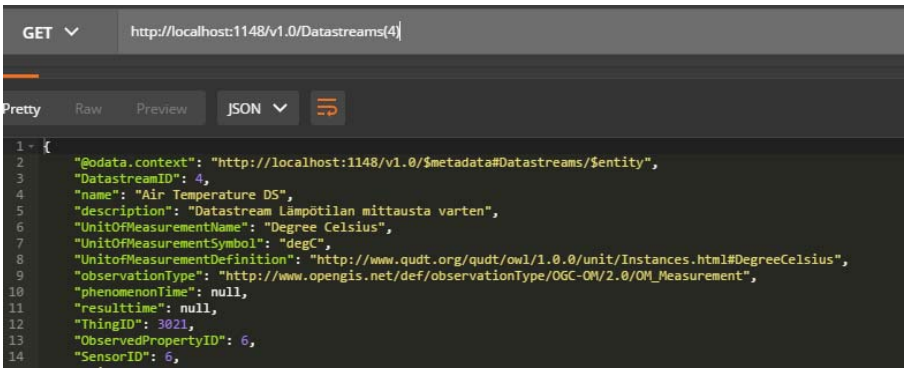
palauttaa myös linkit niihin objekteihin, joihin kyseisellä objektilla on relaatio, mutta tässä sovelluksessa tätä ominaisuutta ei ole toteutettu.



KUVA 14 POST-kutsu useamman objektin luomiseksi kerralla. (Joona Myllys, 2018)



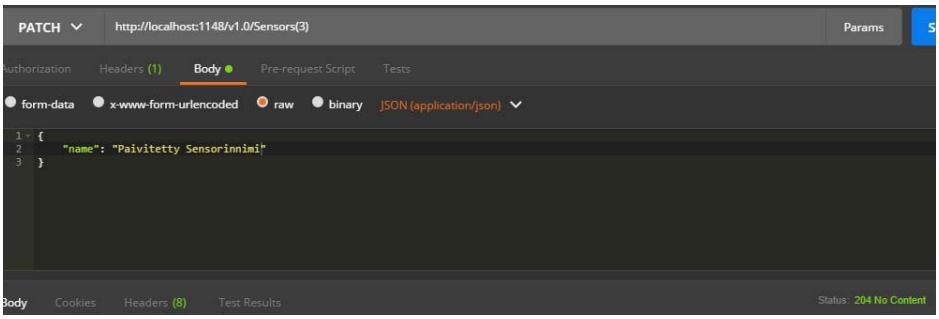
KUVA 15 Juuri luotu uusi Thing-objekti jossa viittaus juuri luotuun Locationiin. (Joona Myllys, 2018)



KUVA 16 Juuri luotu uusi Datastream-objekti jossa viittaukset juuri luotuihin Thingiin, Sensoriin ja Observed-propertyyn. (Joona Myllys, 2018)

8.3 PATCH

PATCH-metodilla korvataan olemassa olevaa dataa tietokannasta, jos esimerkiksi pitää päivittää virheellistä tietoa tai pitää lisätä puuttuva tieto jälkikäteen. PATCH-kutsu toimii siten, että URI:ssa annetaan sen entiteetin nimi ja kyseisen objektin id, jota halutaan muokata, tämän lisäksi body-osassa lähetetään sovellukselle ne tiedot mitä halutaan päivittää. Kuvassa 17 päivitetään Sensorin nimeä objektille, jonka id on 3. Onnistuneesta PATCH-kutsusta sovellus lähettää käyttäjälle HTTP-statuskoodin 204, joka tarkoittaa, että sovellus on toteuttanut kutsun onnistuneesti.

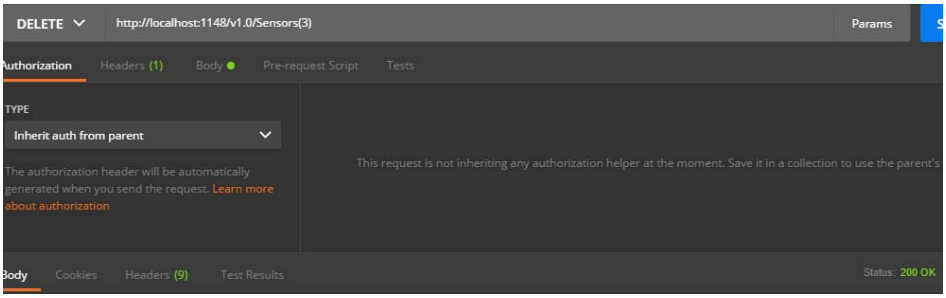


KUVA 17 Sensorin nimen päivitys. (Joona Myllys, 2018)

KUVA 18 Päivitetyn sensorin tiedot päivityksen jälkeen. (Joona Myllys, 2018)

8.4 DELETE

Delete-metodilla nimensä mukaisesti poistetaan tietokannasta dataa. Kutsulla annetaan URI:ssa entiteetin nimi ja poistettavan objektin id, on myös hyvä tietää, että standardin mukaisesti, esimerkiksi, kun poistetaan Thing-objekti, tulee poistaa kaikki Datastream-objektit, joissa viitataan kyseiseen Thing-objektiin. Tämän kaltaisia sidoksia, joissa poistetaan liittyvät objektit, löytyy enemmänkin sovelluksesta ja ne ovat SensorThings API:n standardin mukaisesti toteutettu. Onnistuneesta DELETE-kutsusta, sovellus palauttaa HTTP-koodin 200, eli OK.



KUVA 19 Sensorin jonka id on 3, poistaminen DELETE-kutsulla

8.5 Jatkokehitys

Sovellukseen on toteutettu aivan yksinkertaisimmat perustason toiminnallisuudet. Jatkokehitystä siihen, että sovellus on täysin Sensorthingsin standardia noudattava, vaaditaan vielä mittavat määrät koska, standardin mukaisia ominaisuuksia, joita tässä demosovelluksessa ei tueta, on erittäin paljon. Perustason toiminnallisuuksiin, jotka tässä työssä jäivät toteuttamatta, ovat PATCH-metodiin myös sisempien JSON-objektien lukeminen ja korvaaminen kantaan (kuva 20) ja POST- ja PATCH-metodeissa suoraan linkittäminen olemassaoleviin objekteihin body-osiossa standardin mukaisesti @iot.id muotoisena (kuva 21). OData myös tarjoaa suuren määrän lisäominaisuuksia joita jatkokehityksessä pystyisi lisäämään, kuten mahdollisuus luoda esimerkiksi uusi Location-objekti linkittäen sen olemassaolevaan Thing-objektiin, kertoen sen URI:ssa POST-kutsulla (/v1.0/Things(id)/Locations). OData tarjoaa myös parametreja kuten $expand, $select, joilla käyttäjä pystyy tekemään tarkempia kutsuja sovellukselle, mitä dataa se toivoo sen palauttavan, nämäkin ominaisuudet jäävät jatkokehitykseen.

```
{
  "name": "Temperature Monitoring System",
  "description": "Sensor system monitoring area temperature",
  "properties": {
    "Deployment Condition": "Deployed in a third floor balcony",
    "Case Used": "Radiation shield"
  }
}
```

KUVA 20 Esimerkki sisemmästä objektista JSONin sisällä (Sensorup, 2018)

```
{
  "name": "Temperature Monitoring System",
  "description": "Sensor system monitoring area temperature",
  "properties": {
    "Deployment Condition": "Deployed in a third floor balcony",
    "Case Used": "Radiation shield"
  },
  "Locations": [
    {"@iot.id":1}
  ]
}
```

KUVA 21 Esimerkki body-osiossa tulevasta @iot.id muotoisesta muuttujasta (Sensorup, 2018)

9 YHTEENVETO

Loppujen lopuksi integraatio SaMi-järjestelmään ei onnistunut tässä opinnäytetyössä. SaMi-järjestelmän tietokantarakenne ei ole sopiva suoraa Sensorthingsin integrointia varten, vaan on siihen tehtävä muutoksia, siten ettei olemassa olevat rakenteet ja rajapinnat rikkoudu. Integraation haasteellisuus johti siihen, että toteutettiin nuo aiemmin esitellyt 8 taulua, joiden pohjalta esimerkkisovellusta kehitettiin.

Haasteita työssä tuotti standardin mukaisten ominaisuuksien toteuttaminen, koska niitä piti työstää tekniikoin jotka eivät olleet ennestään tuttuja. Haasteita tuotti myös standardin vaatimien ominaisuuksien määrä ja tarkkuus, jotkut ominaisuudet ovat standardissa kuvattu erittäin tarkasti, miten tulee dataa sisään ja mitä sovelluksen tulisi palauttaa, mutta kaikki tältä väliltä jää kehittäjälle pähkäiltäväksi, joka osoittautui suureksi haasteeksi tarvittavan osaamisen puuttuessa. Haasteiden kanssa taistelusta oppi paljon uusia asioita ohjelmoinnista ja oppi myös eri tavoin hakemaan tietoa ja selvittämään ongelmien juurisyitä.

Kaiken kaikkiaan voidaan sanoa, että tuli opittua paljon uusia asioita ohjelmoinnista, erilaisista tekniikoista ja rajapinnoista tämän opinnäytetyö prosessin aikana. Tuli monta kertaa huomattua, että kantapään kautta oppiminen on erittäin pätevä keino oppia uusia asioita ohjelmoinnista, koska kun tulee ongelmia vastaan, tulee tutkittua asioita paljon laajemmin, jotta löytäisi ongelmaan ratkaisun ja täten tulee vähän jopa huomaamatta tutustuttua asioihin, joihin ei mahdollisesti olisi muuten tutustunut.

LÄHTEET JA TUOTETUT AINEISTOT

MICROSOFT 2017. [Viitattu 2018-3-22] SAATAVISSA: https://www.visualstudio.com/

Sivonen V-M 2004. [Viitattu 2018-3-22] SAATAVISSA: https://www.cs.helsinki.fi/u/pohjalai/k04/ohpe/seminar/Sivonen-CSharp.pdf

MICROSOFT 2018. [VIITATTU 2018-3-22] SAATAVISSA: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms

BRÖRING, A. STASCH, S ja ECHTERHOFF, J. 2012. OGC Sensor Observation Service Interface Standard. [Viitattu 2018-3-25]. Saatavissa: http://www.opengis.net/doc/IS/SOS/2.0

SENSORUP 2016. Comparison of SensorThings API and Sensor Observation Service [Viitattu 2018-3-25]. Saatavissa: https://www.sensorup.com/blog/2016/03/01/comparison-of-sensorthings-api-and-sensor-observation-service/

OGC SensorThings API Part 1: Sensing [Viitattu 2018-3-25] Saatavissa: http://www.opengeospatial.org/standards/sensorthings

LIANG, S. Huang, C. Khalafbeigi, T. OGC SensorThings API Part 1: Sensing [Viitattu 2018-3-25] Saatavissa: http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

ITU Recommendation Y.2060 (06/12) [Viitattu 2018-3-25] Saatavissa: https://www.itu.int/rec/T-REC-Y.2060-201206-I

Logistiikan Maailma, Esineiden Internet [Viitattu 2018-3-25] Saatavissa: http://www.logistiikanmaailma.fi/logistiikka/digitalisaatio/esineiden-internet/

SECLISTS 2018. Hijacking of arbitrary miSafes Mi-Cam video baby monitors [Viitattu 2018-3-25] Saatavissa: http://seclists.org/fulldisclosure/2018/Feb/59

F-Secure, 2018. PINNING DOWN THE IOT [Viitattu 2018-3-25] Saatavissa: https://fsecureconsumer.files.wordpress.com/2018/01/f-secure_pinning-down-the-iot_final.pdf

OASIS 2017. OASIS Open Data Protocol (Odata) TC [Viitattu 2018-3-26] Saatavissa: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata

OData 2017. [Viitattu 2018-3-26] Saatavissa: www.odata.org

Microsoft 2018. JSON data in SQL Server [Viitattu 2018-3-26] Saatavissa: https://docs.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server

Savonia. Avoin tietojärjestelmä kiihdyttämään uusien ympäristömonitoroinnin palveluiden syntymistä [Viitattu 2018-4-19] Saatavissa: http://portal.savonia.fi/amk/fi/tutustu-savoniaan/avoin-tietojarjestelma-kiihdyttamaan-uusien-ymparistomonitoroinnin-palvelujen

EI SAATAVISSA OLEVAT LÄHTEET

Mikko Pääkkönen, Sähköpostiviesti 27.03.2018.